INGENIERÍA DEL SOFTWARE II

PROYECTO – REFACTORIZACIÓN

2025 - 2026

AUTORES:
Ilargi Quetzalli Matilde López
Gontzal Leizabe Escartin

ÍNDICE

Autora: Ilargi

“Write short units of code” (capítulo 2)

Ubicación: DataAccess - createRide()

Código inicial

```
public Ride createRide(String from, String to, Date date, int nPlaces, float price, String driverName)
        throws RideAlreadyExistException, RideMustBeLaterThanTodayException {
    System.out.println(
            ">> DataAccess: createRide=> from= " + from + " to= " + to + " driver=" + driverName + " date " + date);
    if (driverName==null) return null;
    try {
        if (new Date().compareTo(date) > 0) {
            System.out.println("ppppp");
            throw new RideMustBeLaterThanTodayException(

                    ResourceBundle.getBundle("Etiquetas").getString("CreateRideGUI.ErrorRideMustBeLaterThanToday"));
        }

        db.getTransaction().begin();
        Driver driver = db.find(Driver.class, driverName);
        if (driver.doesRideExists(from, to, date)) {
            db.getTransaction().commit();
            throw new RideAlreadyExistException(
                    ResourceBundle.getBundle("Etiquetas").getString("DataAccess.RideAlreadyExist"));
        }
        Ride ride = driver.addRide(from, to, date, nPlaces, price);
        // next instruction can be obviated
        db.persist(driver);
        db.getTransaction().commit();

        return ride;
    } catch (NullPointerException e) {
        // TODO Auto-generated catch block
        return null;
    }


}
```

Código refactorizado

```
public Ride createRide(String from, String to, Date date, int nPlaces, float price, String driverName)
        throws RideAlreadyExistException, RideMustBeLaterThanTodayException {
    System.out.println(
            ">> DataAccess: createRide=> from= " + from + " to= " + to + " driver=" + driverName + " date " + date);
```

```
    if (driverName==null) return null;
    try {
        validateDate(date);
        Ride ride = createRideTransaction(from, to, date, nPlaces, price, driverName);
        return ride;
    } catch (NullPointerException e) {
        // TODO Auto-generated catch block
        return null;
    }
}

private Ride createRideTransaction(String from, String to, Date date, int nPlaces, float price, String driverName)
        throws RideAlreadyExistException {
    db.getTransaction().begin();
    Driver driver = db.find(Driver.class, driverName);
    checkRideExists(from, to, date, driver);
    Ride ride = driver.addRide(from, to, date, nPlaces, price);
    // next instruction can be obviated
    db.persist(driver);
    db.getTransaction().commit();
    return ride;
}
private void checkRideExists(String from, String to, Date date, Driver driver) throws RideAlreadyExistException {
    if (driver.doesRideExists(from, to, date)) {
        db.getTransaction().commit();
        throw new RideAlreadyExistException(
                ResourceBundle.getBundle("Etiquetas").getString("DataAccess.RideAlreadyExist"));
    }
}
private void validateDate(Date date) throws RideMustBeLaterThanTodayException {
    if (new Date().compareTo(date) > 0) {
        System.out.println("ppppp");
        throw new RideMustBeLaterThanTodayException(
                ResourceBundle.getBundle("Etiquetas").getString("CreateRideGUI.ErrorRideMustBeLaterThanToday"));
    }
}
```

Descripción del code smell detectado y refactorización realizada

El método original *createRide()* presentaba un *bad smell* de tipo *Long Method* ya que tenía 26 líneas de código, superando el límite de 15 líneas. Para corregirlo, se ha utilizado *Extract Method*, dividiendo el código en 3 métodos auxiliares más pequeños.
Tras la refactorización, el método *createRide()* ha quedado con un total de 13 líneas de código, eliminando así el *bad smell* y consiguiendo una mejora de la mantenibilidad, ya que las unidades pequeñas son fáciles de entender, probar y reutilizar.
Los métodos extraídos son los siguientes:

- `private void checkRideExists(String from, String to, Date date, Driver driver) throws RideAlreadyExistException`

Verifica si el conductor tiene ya registrado un viaje con el mismo origen, destino y fecha. Si el viaje existe, lanza una excepción.

- `private void validateDate(Date date) throws RideMustBeLaterThanTodayException`

  Controla que la fecha dada sea posterior a la fecha actual.

- `private Ride createRideTransaction(String from, String to, Date date, int nPlaces, float price, String driverName)`

  Gestiona la transacción de la base de datos para crear el nuevo viaje.

"Write simple units of code" (capítulo 3)

Ubicación: DataAccess - bookRide()

Código inicial

```
public boolean bookRide(String username, Ride ride, int seats, double desk) {
    try {
        db.getTransaction().begin();
        Traveler traveler = getTraveler(username);
        if (traveler == null) {
            return false;
        }
        if (ride.getnPlaces() < seats) {
            return false;
        }
        double ridePriceDesk = (ride.getPrice() - desk) * seats;
        double availableBalance = traveler.getMoney();
        if (availableBalance < ridePriceDesk) {
            return false;
        }
        Booking booking = new Booking(ride, traveler, seats);
        booking.setTraveler(traveler);
        booking.setDeskontua(desk);
        db.persist(booking);
        ride.setnPlaces(ride.getnPlaces() - seats);
        traveler.addBookedRide(booking);
        traveler.setMoney(availableBalance - ridePriceDesk);
        traveler.setIzoztatutakoDirua(traveler.getIzoztatutakoDirua() + ridePriceDesk);
        db.merge(ride);
        db.merge(traveler);
        db.getTransaction().commit();
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
```

```
    }
}
```

Código refactorizado

```
public boolean bookRide(String username, Ride ride, int seats, double desk) {
    try {
        db.getTransaction().begin();
        boolean valido = true;
        Traveler traveler = getTraveler(username);
        valido = travelerAndSeatsValid(ride, seats, valido, traveler);
        double ridePriceDesk = (ride.getPrice() - desk) * seats;
        double availableBalance = traveler.getMoney();
        if (availableBalance < ridePriceDesk) {
            valido = false;
        }
        if(!valido) return false;
        Booking booking = new Booking(ride, traveler, seats);
        booking.setTraveler(traveler);
        booking.setDeskontua(desk);
        db.persist(booking);
        ride.setnPlaces(ride.getnPlaces() - seats);
        traveler.addBookedRide(booking);
        traveler.setMoney(availableBalance - ridePriceDesk);
        traveler.setIzoztatutakoDirua(traveler.getIzoztatutakoDirua() + ridePriceDesk);
        db.merge(ride);
        db.merge(traveler);
        db.getTransaction().commit();
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
private boolean travelerAndSeatsValid(Ride ride, int seats, boolean valido, Traveler traveler) {
    if (traveler == null) {
        valido = false;
    }
    if (ride.getnPlaces() < seats) {
        valido = false;
    }
    return valido;
}
```

Descripción del code smell detectado y refactorización realizada

El método original *bookRide()* presentaba un *bad smell* de tipo alta complejidad ya que tenía una complejidad ciclomática de 5, superando el límite de 4. Para corregirlo, se ha

utilizado *Extract Method*, dividiendo el código en un método auxiliar más pequeño. Tras la refactorización, el método *bookRide()* ha quedado con una complejidad ciclomática de 4, consiguiendo así mantener un número bajo de puntos de ramificación.
El método extraído es el siguiente:

- `private boolean travelerAndSeatsValid(Ride ride, int seats, boolean valido, Traveler traveler)`

  Verifica que el viajero exista en la base de datos y que el viaje tenga asientos disponibles antes de continuar con la reserva.

"Duplicate code" (capítulo 4)

Ubicación: DataAccess

Código inicial

```
147         Movement m1 = new Movement(traveler1, "BookFreeze", 20);
148         Movement m2 = new Movement(traveler1, "BookFreeze", 40);
149         Movement m3 = new Movement(traveler1, "BookFreeze", 5);
150         Movement m4 = new Movement(traveler2, "BookFreeze", 4);
151         Movement m5 = new Movement(traveler1, "BookFreeze", 3);
152         Movement m6 = new Movement(driver1, "Deposit", 15);
153         Movement m7 = new Movement(traveler1, "Deposit", 168);
```

Código refactorizado

```
29      private static final String bookFreeze = "BookFreeze";
```

```
147         Movement m1 = new Movement(traveler1, bookFreeze, 20);
148         Movement m2 = new Movement(traveler1, bookFreeze, 40);
149         Movement m3 = new Movement(traveler1, bookFreeze, 5);
150         Movement m4 = new Movement(traveler2, bookFreeze, 4);
151         Movement m5 = new Movement(traveler1, bookFreeze, 3);
152         Movement m6 = new Movement(driver1, "Deposit", 15);
153         Movement m7 = new Movement(traveler1, "Deposit", 168);
```

Descripción del code smell detectado y refactorización realizada

El código original presentaba un *bad smell* de tipo *Duplicate code* ya que se repetía el *String "BookFreeze"* un total de 5 veces. Para corregirlo se ha utilizado *Extract Constant* extrayendo el valor repetido a una constante estática en la clase y sustituyendo las apariciones del *String* por el uso de dicha constante. Tras realizar la refactorización, el código se vuelve más claro y fácil de mantener, evitando la duplicación.

“Keep unit interfaces small” (capítulo 5)

Ubicación: DataAccess y BLFacadeImplementation - erreklamazioaBidali()

Código inicial

DataAccess

```
public boolean erreklamazioaBidali(String nor, String nori, Date gaur, Booking booking, String textua,
        boolean aurk) {
    try {
        db.getTransaction().begin();

        Complaint erreklamazioa = new Complaint(nor, nori, gaur, booking, textua, aurk);
        db.persist(erreklamazioa);
        db.getTransaction().commit();
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
```

BLFacadeImplementation

```
@Override
   public boolean erreklamazioaBidali(String nor, String nori, Date gaur, Booking book, String textua,
boolean aurk) {
    dbManager.open();
    boolean sent = dbManager.erreklamazioaBidali(nor, nori, gaur, book, textua, aurk);
    dbManager.close();
    return sent;
  }
```

Código refactorizado

DataAccess

```
public boolean erreklamazioaBidali(Complaint reclamacion) {
    try {
        db.getTransaction().begin();
        db.persist(reclamacion);
        db.getTransaction().commit();
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
 }
```

BLFacadeImplementation

```
@Override
   public boolean erreklamazioaBidali(String nor, String nori, Date gaur, Booking book, String textua,
boolean aurk) {
      dbManager.open();
      Complaint reclamacion = new Complaint(nor, nori, gaur, book, textua, aurk);
      boolean sent = dbManager.erreklamazioaBidali(reclamacion);
      dbManager.close();
      return sent;
   }
```

Descripción del code smell detectado y refactorización realizada

El método original *erreklamazioaBidali()* presentaba un bad smell de tipo *Long Parameter List*, ya que el método recibía un total de 6 parámetros, superando el límite de 4 parámetros. Para corregirlo, se ha utilizado *Change Method Signature* eliminando los 6 parámetros del método e incorporando un único parámetro de tipo *Complaint*, que agrupa los datos necesarios para la operación. Tras realizar esta refactorización, se ha tenido que modificar además el método *erreklamazioaBidali()* en la clase *BLFacadeImplementation* sustituyendo la llamada *dbManager.erreklamazioaBidali(nor, nori, gaur, book, textua, aurk);* por *dbManager.erreklamazioaBidali(reclamacion);* donde reclamacion es un *Complaint* de creado previamente con los datos correspondientes.

Autor: Gontzal

“Write short units of code” (capítulo 2)

Ubicación: DataAccess

Código inicial

```
public boolean bookRide(String username, Ride ride, int seats, double desk)
{
		try {
			db.getTransaction().begin();
			Traveler traveler = getTraveler(username);
			if (traveler == null) {
				return false;
			}
			if (ride.getnPlaces() < seats) {
				return false;
			}
			double ridePriceDesk = (ride.getPrice() - desk) * seats;
			double availableBalance = traveler.getMoney();
			if (availableBalance < ridePriceDesk) {
				return false;
			}
			Booking booking = new Booking(ride, traveler, seats);
			booking.setTraveler(traveler);
			booking.setDeskontua(desk);
			db.persist(booking);
			ride.setnPlaces(ride.getnPlaces() - seats);
			traveler.addBookedRide(booking);
			traveler.setMoney(availableBalance - ridePriceDesk);

traveler.setIzoztatutakoDirua(traveler.getIzoztatutakoDirua() +
ridePriceDesk);
			db.merge(ride);
			db.merge(traveler);
			db.getTransaction().commit();
			return true;
		} catch (Exception e) {
			e.printStackTrace();
			db.getTransaction().rollback();
			return false;
		}
	}
```

Código refactorizado

```
	public boolean bookRide(String username, Ride ride, int seats, double
desk) {
		try {
		Traveler t = getTraveler(username);
		Ride r = ride;
		int s = seats;
		double d = desk;
		db.getTransaction().begin();
		validateSufficientFunds(t, r, s);
		validateSeatAvailability(r, s);
		processPayment(t, r.getDriver(), r.getPrice(), s);
		persistBooking(r, t, s);
		db.getTransaction().commit();
		return true;
		} catch (Exception e) {
			e.printStackTrace();
			db.getTransaction().rollback();
			return false;
		}
	}

	public void validateSufficientFunds(Traveler t, Ride r, int s) {
		double total = r.getPrice()*s;
		if(t.getMoney()<total) {
			db.getTransaction().commit();
		  throw new IllegalArgumentException("Diru gutxi");
		}
	}

	private void validateSeatAvailability(Ride ride, int seats) {
		if (ride.getnPlaces() < seats) {
			db.getTransaction().commit();
			throw new IllegalArgumentException("Plaza gutxi");
		}
	}

	private void processPayment(Traveler traveler, Driver driver, double
price, int seats) {
		double total = price * seats;
		traveler.setMoney(traveler.getMoney() - total);
		driver.setMoney(driver.getMoney() + total);
	}

	private void persistBooking(Ride ride, Traveler traveler, int seats) {
		Booking booking = new Booking(ride, traveler, seats);
		traveler.addBookedRide(booking);
		db.persist(booking);
	}
```

Descripción del code smell detectado y refactorización realizada

El método original *bookRide()* presentaba un *bad smell* de tipo *Long Method* ya que tenía 32 líneas de código, superando el límite de 15 líneas. Para corregirlo, se ha utilizado *Extract Method*, dividiendo el código en 4 métodos auxiliares más pequeños. Se ha acortado el metodo bookRide y se han creado los siguientes subprocesos que lo forman:

```
public void validateSufficientFunds(Traveler t, Ride r, int s)

private void validateSeatAvailability(Ride ride, int seats)

private void processPayment(Traveler traveler, Driver driver, double price,
int seats)

private void persistBooking(Ride ride, Traveler traveler, int seats)
```

“Write simple units of code” (capítulo 3)

Ubicación:DataAccess

Código inicial

```
	public void cancelRide(Ride ride) {
		try {
			db.getTransaction().begin();
			for (Booking booking : ride.getBookings()) {
				if (booking.getStatus().equals("Accepted") ||
booking.getStatus().equals("NotDefined")) {
					double price = booking.prezioaKalkulatu();
					Traveler traveler = booking.getTraveler();
					double frozenMoney =
traveler.getIzoztatutakoDirua();
					traveler.setIzoztatutakoDirua(frozenMoney -
price);
					double money = traveler.getMoney();
					traveler.setMoney(money + price);
					db.merge(traveler);
					db.getTransaction().commit();
					addMovement(traveler, "BookDeny", price);
					db.getTransaction().begin();
				}
				booking.setStatus("Rejected");
				db.merge(booking);
			}
			ride.setActive(false);
			db.merge(ride);
			db.getTransaction().commit();
		} catch (Exception e) {
			if (db.getTransaction().isActive()) {
```

```
                        db.getTransaction().rollback();
                    }
                    e.printStackTrace();
                }
            }
```

Código refactorizado

```
        public void cancelRide(Ride ride) {
            try {
                db.getTransaction().begin();
                for (Booking booking : ride.getBookings()) {
                    diruaItzuli(booking);
                }
                ride.setActive(false);
                db.merge(ride);
                db.getTransaction().commit();
            } catch (Exception e) {
                if (db.getTransaction().isActive()) {
                    db.getTransaction().rollback();
                }
                e.printStackTrace();
            }
        }
        public void diruaItzuli(Booking b) {
            if (b.getStatus().equals("Accepted") ||
b.getStatus().equals("NotDefined")) {
                double price = b.prezioaKalkulatu();
                Traveler traveler = b.getTraveler();
                double frozenMoney = traveler.getIzoztatutakoDirua();
                traveler.setIzoztatutakoDirua(frozenMoney - price);
                double money = traveler.getMoney();
                traveler.setMoney(money + price);
                db.merge(traveler);
                db.getTransaction().commit();
                addMovement(traveler, "BookDeny", price);
                db.getTransaction().begin();
            }
            b.setStatus("Rejected");
            db.merge(b);
        }
```

Descripción del code smell detectado y refactorización realizada

Descripción del code smell detectado y refactorización realizada

El método original *cancelRide()* presentaba un *bad smell* de tipo alta complejidad ya que tenía una complejidad ciclomática de 5, superando el límite de 4. Para corregirlo, se ha utilizado *Extract Method*, dividiendo el código en un método auxiliar más pequeño. Tras la refactorizacion, se ha creado el metodo diruaItzuli() y se ha conseguido una complejidad ciclomatica de 4

El metodo extraido es

```
public void diruaItzuli(Booking b)
```

“Duplicate code” (capítulo 4)

Ubicación:

Código inicial

```
driver1.addRide("Donostia", madrid, date2, 5, 20);
driver1.addRide("Irun", "Donostia", date2, 5, 2);
driver1.addRide(madrid, "Donostia", date3, 5, 5);
driver1.addRide("Barcelona", madrid, date4, 0, 10);
driver2.addRide("Donostia", "Hondarribi", date1, 5, 3);
```

Código refactorizado

```
private static final String donostia= "Donostia";

driver1.addRide(donostia, madrid, date2, 5, 20);
driver1.addRide("Irun", donostia, date2, 5, 2);
driver1.addRide(madrid, donostia, date3, 5, 5);
driver1.addRide("Barcelona", madrid, date4, 0, 10);
driver2.addRide(donostia, "Hondarribi", date1, 5, 3);
```

Descripción del code smell detectado y refactorización realizada

El código original presentaba un *bad smell* de tipo *Duplicate code* ya que se repetía el *String “BookFreeze”* un total de 4 veces. Para corregirlo se ha utilizado *Extract Constant* extrayendo el valor repetido a una constante estática en la clase y sustituyendo las apariciones del *String* por el uso de dicha constante. Tras realizar la refactorización, el código se vuelve más claro y fácil de mantener, evitando la duplicación.

“Keep unit interfaces small” (capítulo 5)

Ubicación: DataAccess

Código inicial

```
	public Ride createRide(String from, String to, Date date, int nPlaces,
float price, String driverName)
			throws RideAlreadyExistException,
RideMustBeLaterThanTodayException {
		System.out.println(
				">> DataAccess: createRide=> from= " + from + " to=
" + to + " driver=" + driverName + " date " + date);
		if (driverName==null) return null;
		try {
			if (new Date().compareTo(date) > 0) {
				System.out.println("ppppp");
				throw new RideMustBeLaterThanTodayException(

ResourceBundle.getBundle("Etiquetas").getString("CreateRideGUI.ErrorRideMustB
eLaterThanToday"));
			}
			db.getTransaction().begin();
			Driver driver = db.find(Driver.class, driverName);
			if (driver.doesRideExists(from, to, date)) {
				db.getTransaction().commit();
				throw new RideAlreadyExistException(

ResourceBundle.getBundle("Etiquetas").getString("DataAccess.RideAlreadyExist"
));
			}
			Ride ride = driver.addRide(from, to, date, nPlaces,
price);
			// next instruction can be obviated
			db.persist(driver);
			db.getTransaction().commit();
			return ride;
		} catch (NullPointerException e) {
			// TODO Auto-generated catch block
			return null;
		}

	}
```

Código refactorizado

```
public Ride createRide(RideRequest r)
			throws RideAlreadyExistException,
RideMustBeLaterThanTodayException {
```

```
            System.out.println(
                        ">> DataAccess: createRide=> from= " + r.getFrom() +
" to= " + r.getTo() + " driver=" + r.getDriverName() + " date " +
r.getDate());
            if (r.getDriverName()==null) return null;
            try {
                  if (new Date().compareTo(r.getDate()) > 0) {
                        System.out.println("ppppp");
                        throw new RideMustBeLaterThanTodayException(

ResourceBundle.getBundle("Etiquetas").getString("CreateRideGUI.ErrorRideMustB
eLaterThanToday"));
                  }
                  db.getTransaction().begin();
                  Driver driver = db.find(Driver.class, r.getDriverName());
                  if (driver.doesRideExists(r.getFrom(), r.getTo(),
r.getDate())) {
                        db.getTransaction().commit();
                        throw new RideAlreadyExistException(

ResourceBundle.getBundle("Etiquetas").getString("DataAccess.RideAlreadyExist"
));
                  }
                  Ride ride = driver.addRide(r.getFrom(), r.getTo(),
r.getDate(), r.getnPlaces(), r.getPrice());
                  // next instruction can be obviated
                  db.persist(driver);
                  db.getTransaction().commit();
                  return ride;
            } catch (NullPointerException e) {
                  // TODO Auto-generated catch block
                  return null;
            }
```

```
public class RideRequest {
   private String from;
   private String to;
   private Date date;
   private int nPlaces;
   private float price;
   private String driverName;
      public String getFrom() {
            return from;
      }
      public void setFrom(String from) {
            this.from = from;
      }
      public String getTo() {
            return to;
      }
      public void setTo(String to) {
            this.to = to;
```

```
	}
	public Date getDate() {
		return date;
	}
	public void setDate(Date date) {
		this.date = date;
	}
	public int getnPlaces() {
		return nPlaces;
	}
	public void setnPlaces(int nPlaces) {
		this.nPlaces = nPlaces;
	}
	public float getPrice() {
		return price;
	}
	public void setPrice(float price) {
		this.price = price;
	}
	public String getDriverName() {
		return driverName;
	}
	public void setDriverName(String driverName) {
		this.driverName = driverName;
	}
}
```

Descripción del code smell detectado y refactorización realizada

El método original *createRide()* presentaba un bad smell de tipo *Long Parameter List*, ya que el método recibía un total de 6 parámetros, superando el límite de 4 parámetros. Para corregirlo, se ha utilizado *Change Method Signature* eliminando los 6 parámetros del método e incorporando un único parámetro de tipo *RideRequest*, que agrupa los datos necesarios para la operación.